Centre de recherche sur les transports (C.R.T.)
Centre for Research on Transportation
CRT-2003-07

THE SINGLE-NODE DYNAMIC SERVICE SCHEDULING AND DISPATCHING PROBLEM

by

Leonardo Campo Dall'Orto
Teodor Gabriel Crainic
José Eugenio Leal
Warren B. Powell

February 2003

Université de Montréal
C.P. 6128, succursale Centre-ville, Montréal QC H3C 3J7 Canada
Téléphone: (514) 343-7575 / Télécopieur: (514) 343-7121

crt@crt.umontreal.ca
www.crt.umontreal.ca

HEC ÉCOLE POLYTECHNIQUE MONTRÉAL Université de Montréal

103-07285/

PUBLICATION
CRT-2003-07

THE SINGLE-NODE DYNAMIC SERVICE SCHEDULING AND DISPATCHING PROBLEM

by

Leonardo Campo Dall'Orto[1]
Teodor Gabriel Crainic[2]
José Eugenio Leal[3]
Warren B. Powell[4]

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) and by the Fonds pour la Formation de Chercheurs et l'Aide à la Recherche (FCAR).

1. Vale do Rio Doce and Department of Industrial Engineering, Pontificia Universidade Catolica do Rio de Janeiro, Rua Marquês de São Vicente, 225, Gávea, Cx. Postal: 38097, Rio de Janeiro, RJ 22453-900, Brazil
2. Département de management et technologie, Université du Québec à Montréal, Montréal, Canada H3C 3J7 and Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Canada H3C 3J7
3. Department of Industrial Engineering, Pontificia Universidade Catolica do Rio de Janeiro, Rua Marquês de São Vicente, 225, Gávea, Cx. Postal: 38097, Rio de Janeiro, RJ 22453-900, Brazil
4. Department of Civil Engineering and Operations Research, School of Engineering and Applied Science, Princeton University, Princeton, NJ 08544, U.S.A.

Dépôt légal – Bibliothèque nationale du Québec
Bibliothèque nationale du Canada, 2003

Abstract

In this paper, we focus on a particular version of the Dynamic Service Network Design (DSND) problem, namely the case of a single-terminal that dispatches services to a number of customers and other terminals. We present a time-dependent, stochastic formulation that aims to optimize the problem over a given planning horizon and propose a solution approach based on dynamic programming principles. We also present a static, single-period, formulation of the single node problem that appears as a subproblem when addressing the time-dependent version and general service network design cases. Despite is apparent simplicity, it is still a network design problem and exact solution methods are not sufficiently fast. We therefore propose two tabu search meta-heuristics based on the ejection chains concept. We also introduce a learning mechanism that takes advantage of experiences gathered in repeated executions. Experiments with problem instances derived from real cases indicate that the proposed solution methods are efficient and yield good solutions.
Keywords: dynamic service network design, stochastic formulation, resource allocation, tabu search, ejection chains.

Résumé

Dans cet article, nous examinons un cas particulier du problème de conception dynamique du réseau de service : le cas d'un terminal qui envoie des services de transport vers des clients ou d'autres terminaux. Nous présentons une formulation dynamique et stochastique qui vise à trouver une solution optimale au problème pour l'ensemble de l'horizon de planification, ainsi qu'une approche de résolution basée sur des principes de programmation dynamique. Nous examinons également la formulation statique du problème, définie pour une seule période, qui apparaît comme sous-problème dans le modèle dynamique, ainsi que lors de la résolution de problèmes généraux de conception de réseaux dynamiques. En dépit de son apparente simplicité, ce dernier problème appartient à la classe de formulations de conception de réseau et les méthodes exactes de résolution n'offrent pas de performances satisfaisantes. Nous proposons donc des méta-heuristiques de recherche avec tabous, basées sur des chaînes d'éjection. Nous proposons également un mécanisme d'apprentissage qui accumule l'information au cours d'exécutions répétées. Les résultats d'expérimentations à l'aide de cas dérivés de données réelles indiquent que les méthodes proposées sont efficaces et fournissent des solutions de bonne qualité.
Mots clefs : conception de réseau de service dynamique, formulation stochastique, allocation de ressources, recherches avec tabous, chaînes d'éjection.

Introduction

Less-than-truckload (LTL) motor carriers, railways, and ocean container lines handle the flows of small shipments by consolidating them on common vehicles that must be dispatched over time. In the case of LTL carriers (and to a lesser extent rail), it is necessary to dynamically dispatch vehicles to move shipments that arrive according to a potentially random, and nonstationary, process. The decision to dispatch a vehicle must trade off the cost of holding freight that has already arrived against the cost of dispatching a vehicle that is not quite full. When these dispatches occur in the context of larger networks, we must also consider the cost of moving shipments from the destination of the vehicle to the final destination of each shipment.

Service Network Design (*SND*) is often associated with the class of problems that address the issue of determining the set of services and characteristics that optimize these goals. *Dynamic*, or time-dependent, *Service Network Design* (*DSND*) arises when the schedule is a function of time. SND problems are NP-Hard and are generally represented as mixed-integer multi-commodity, capacitated, network optimization formulations (Magnanti and Wong 1986; Minoux 1986; Balachrishnan, Magnanti, and Mirchandani 1997; Crainic 2000). Crainic (2000, 2003; see also Crainic and Laporte 1997) provides a general view of these problems and surveys the main methodological contributions to the area, while Cordeau, Toth, and Vigo (1998) present a review of the most recent contributions dealing with train routing and scheduling with regard to both freight and passenger transportation. These problems exhibit very weak LP relaxations, which cripple any algorithms that depend on branch and bound. As a result, the field is dominated by various heuristics (e.g., Barnhart, Jin, and Vance 2000; Barnhart and Schneur 1996; Kim *et al.* 1999; Crainic, Ferland, and Rousseau 1984; Crainic and Rousseau 1986; Crainic and Roy 1988; Grünert and Sebastian 2000; Powell 1986; Powell and Sheffi, 1989). Dynamic network design problems are even harder, and as a result have received even less attention (e.g., Haghani 1989; Farvolden and Powell 1994; Equi *et al.* 1997; Gorman 1998). To our knowledge, stochastic versions of the network design problem have not even been addressed in the literature.

There have been separate lines of investigation into special cases of the network design problem. There is a history of research into the single link problem, which involves modeling the control of a batch dispatch process of a vehicle from one node to another. Kosten (1967, 1973) was the first to consider the control of a single server stochastic, batch service problem in steady state. Ignall and Kolesar (1972), Deb and Serfozo (1973), and Powell and Humblet (1986) made further contributions to this problem class. Speranza and Ukovich (1992, 1994, and 1996) present deterministic formulations of the single link problem. Burns *et al.* (1984), Blumenfeld *et al.* (1985), and Daganzo (1991) investigate properties of the single link problem in the context of strategies in shipper logistics. Bertazzi and Speranza (1999) use approximate dynamic programming methods to solve a discrete state version of the stochastic problem with a small number of product types. Bertazzi and Speranza (2002) present shipping strategies for several products on a capacitated link, such that the sum of transportation and inventory costs is minimized; The discrete and continuous transport frequencies cases are derived from the general analysis framework. Papadaki and Powell (2002) exploit the monotone structure of the value function to estimate a discrete version of the value function for a single commodity. Papadaki and Powell (2003) introduce linear approximations and nonlinear, concave approximations and show that they produce much faster convergence and can be easily applied to problems with a large number of shipment types.

We propose to extend the basic strategy used in Papadaki and Powell (2003) to the problem of a single node, with vehicles departing over a set of outbound links to different terminals. We assume that there is a set of shipments waiting to be dispatched to various destinations that might serve as intermediate transfer points. Thus, the destinations of the vehicles will generally be a small subset of the set of destinations of shipments. This also introduces the dimension that we have to decide which outbound vehicle a shipment should be assigned to, recognizing that there are both immediate and downstream costs to be considered in such a decision. Our strategy is to formulate a precise version of the problem faced at a single node, using approximations of downstream costs. This view is a realistic model of the dispatching process of large less-than-truckload networks, where terminal managers make decisions using a high level of information about their own terminal (at the point in time that a decision is made), with approximations of downstream impacts (the cost of sending a shipment with final destination k on a truck whose destination is j). This paper directly extends the line of research on single link problems, and represents, we believe, a potential stepping stone to a strategy that can be applied to large-scale networks.

A realistic dispatch problem from a single node might involve dispatching trucks to several dozen destinations. Given the relative ineffectiveness of LP relaxations in branch and bound algorithms for this problem class, it is computationally intractable to formulate even deterministic versions of this problem over multiple time periods. Our strategy is to solve the single-node, multiple time period problem using approximate dynamic programming methods, where the impact of decisions now on the future is approximated using linear approximations. The use of continuous approximations for value functions has been studied for years (see, for example, Tsitsiklis and van Roy 1997; Pflug 1996) but only recently has been recognized for its ability to allow dynamic programming techniques to be applied to high dimensional problems (Powell and Carvalho 1997, 1998; Powell and Topaloglu 2003), most recently in this specific problem class (Papadaki and Powell 2003).

Even with these techniques, the single period problem (which must be solved very quickly) remains computationally challenging. A single node at a single point in time may face several dozen outbound links, creating an integer programming problem with several dozen integer variables. Without a reasonable bound, branch and bound would virtually enumerate the entire tree. Our approximation strategy requires that many of these problems be solved, so speed is essential. We therefore propose a tabu search meta-heuristic (Glover 1989, Glover and Laguna 1997) based on the concept of ejection chains. We also introduce a learning mechanism that goes beyond the usual within-the-search-trajectory memories to take advantage of experiences gathered in repeated executions in actual practice. Experiments with problem instances derived from real cases indicate that the proposed solution methods are efficient and yield good solutions.

The paper is organized as follows: Section 1 presents the problem and introduces the time-dependent formulations and the solution algorithm. Section 2 describes the single-period formulations, the tabu search meta-heuristic, and the long-term learning mechanism. Section 3 presents the experimental setting and the result analysis. We conclude in Section 4.

1 THE SINGLE-NODE MULTIPERIOD SERVICE NETWORK DESIGN PROBLEM

In the single-node-service design problem, shipments must be sent from a terminal to customers using the services available at the terminal. Services could be direct or not. Direct services link the

origin directly to the destination, while the others use intermediate terminals to consolidate cargo. A "*fixed*" cost must be paid to operate a given service (to dispatch a vehicle), independently of the actual load. Transportation costs proportional to the shipments carried may also be involved. Shipments can be kept at the terminal, incurring a holding cost that reflects the service requirements of each individual shipment. We must make the decision at time t to determine whether or not to dispatch a vehicle to each of a set of intermediate transfer points, and which shipments should go on each vehicle. A shipment that is put on a vehicle that takes it to an intermediate transfer terminal also incurs a cost to move the shipment from the intermediate transfer terminal to the final destination. For this paper, this downstream cost is assumed known. Our goal is to minimize the sum of dispatch costs, holding costs at the origin terminal, and the downstream costs for shipments arriving at an intermediate terminal. Costs are minimized over a finite planning horizon.

When the scheduling (or the dispatching) of services is contemplated, a *time* dimension must be introduced in the DSND formulations. This is usually achieved by representing the operations over a certain number of time periods by using a *space-time* network. This increases the size of the problem and also makes it more computationally complex, since the model must incorporate the impact of decisions taken in one period on decisions taken in later periods.

The general approach is as follows. The representation of the physical network, of terminals in particular, is replicated in each period. Services are generated on temporal links. Starting from its origin in a given period, a service arrives (and leaves, in the case of intermediary stops) later at other terminals. Each service leg thus represents a service link between two different terminals at two different time periods. Links between nodes representing the same terminal at two consecutive periods are used to model holding decisions. Figure 1 illustrates this concept through a network with five terminals and several services offered in each terminal. Services S1, S2, and S3 originate at terminal A. Services S1 and S2 are direct, while S3 uses terminal C as an intermediate terminal for consolidation operations. Similarly, service S4 initiates at terminal B, S5 from C, and S6 from D. Since terminals and services are replicated over time, the departing period is indicated (e.g., service S1,1 stands for S1 leaving terminal A at time 1 and arriving at terminal D at period 3). The option of holding the freight at the terminal is illustrated by the gray links that connect the same terminal in subsequent periods of time.

Two types of decision variables are usually defined. Integer design variables are associated with each service. Restricted to {0, 1} values, these variables indicate whether or not the service will be offered at the specified time. When several departures may take place in the same period, either general (nonnegative) integer variables are used or the length of the period is redefined. Continuous variables are used to represent the distribution of freight flows through this service network. Additional decision variables may be introduced to represent the flow of empty vehicles in order to ensure the balance of vehicle flows at terminals. The resulting mixed-integer formulation minimizes the sum of service fixed costs, holding costs, moving loaded and empty vehicles. It considers constraints such as flow conservation at nodes, capacity on service links and in terminals, and general routing and frequency constraints. The formulation we present in this paper follows these general principles.

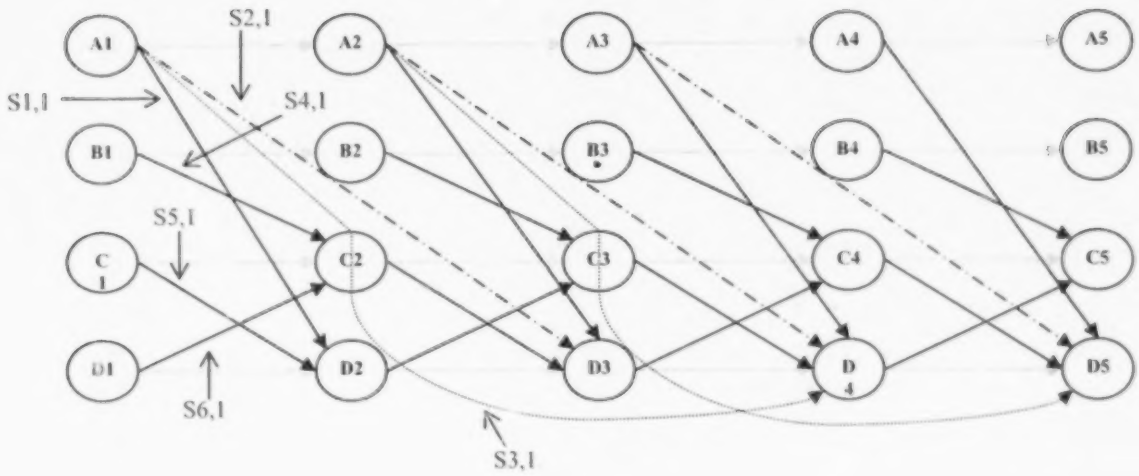

Figure 1 — A Dynamic Service Network

The single-node multiperiod service network design problem may be viewed as a special case of the general dynamic case with only one source node. Although this setting appears to simplify both the formulation and the solution methods, the problem still exhibits strong combinatorial features and most difficulties related to solving network design problems are also encountered in this case. To simplify the presentation, we describe the problem and models in terms of less-than-truckload motor carrier transportation. The developments are general, however, and may be applied to any consolidation transportation case.

The problem represents the decision of a dispatcher of a truck terminal at a given moment. The dispatcher has to determine the best strategy to move freight from the terminal to its destination, selecting between i) sending in the current period, either directly or by using some intermediate terminals, or ii) holding the freight in inventory for shipment at a later period. A fixed cost is incurred when a service is selected (following a dispatching decision). Each service has a finite capacity representing the maximum load the service can carry. Holding costs apply to freight kept for future periods.

We present both a deterministic and a stochastic formulation of the problem. An algorithm is also described for the latter. The two formulations are experimentally contrasted in Section 3.

1.1 The deterministic model

The problem is illustrated in Figure 2 for periods $t = 1,\ldots,T$. All shipments are waiting at a single source node (which is not indexed to simplify notation). Let:

K_t = The set of destination nodes for shipments waiting at time t.

$\hat{R}_{kt}$ = The number of new arrivals of shipments with destination k arriving to the system at time t.

$\hat{R}_t = \left(\hat{R}_{kt}\right)_{k \in K_t}$

We define our state variable using:

R_{kt} = The number of shipments with destination k remaining in the system at the end of period t, after the vehicles have departed..

$R_t = \left(R_{kt}\right)_{k \in K_t}$

The vector R_t is known as an *incomplete* state variable, which plays an important role in the development of approximations for *stochastic versions* of the problem. R_t is also the vector of shipments available at the beginning of time $t+1$ before new arrivals are added in. We index the variable by t (instead of $t+1$) because, in the stochastic version, it indicates the *information content* of the variable. We adopt this version of the state variable here to maintain consistency. The total number of shipments that need to be moved in period t is given by the *complete* state variable, which we write:

$$R_t^+ = R_{t-1} + \hat{R}_t$$

Shipments may be moved either directly to their final destinations or to an intermediary terminal, which will then be responsible for the next dispatch. A *service* is a direct movement of a vehicle from the source to another terminal. Let:

J_t = The set of destinations to which vehicles may be sent at time t.

The decision variables are given by:

$$y_{jt} = \begin{cases} 1 & \text{If a vehicle is dispatched from the source node to node j, departing at time t.} \\ 0 & \text{Otherwise} \end{cases}$$

x_{jkt} = The number of shipments with final destination k to be moved on a vehicle going to terminal j.

Once a shipment has arrived at the transfer terminal j, we then approximate the cost of the remainder of the trip by a linear approximation. Shipments that are not dispatched to a terminal in J_t are held. The costs we wish to minimize are given by:

f_{jt} = The fixed cost of sending a vehicle to destination j at time t.

g_{jkt} = The unit cost of moving a shipment from intermediate terminal j to final destination k when it is dispatched to j at time t.

h_{kt} = The unit cost of holding a shipment with ultimate destination k at the origin node.

The holding costs h_{kt} can be used to capture differences in the value of the shipments.

The inventory equation can now be written as:

$$R_{kt} = R_{k,t-1} + \hat{R}_{k,t} - \sum_{j \in J_t} x_{jkt}$$

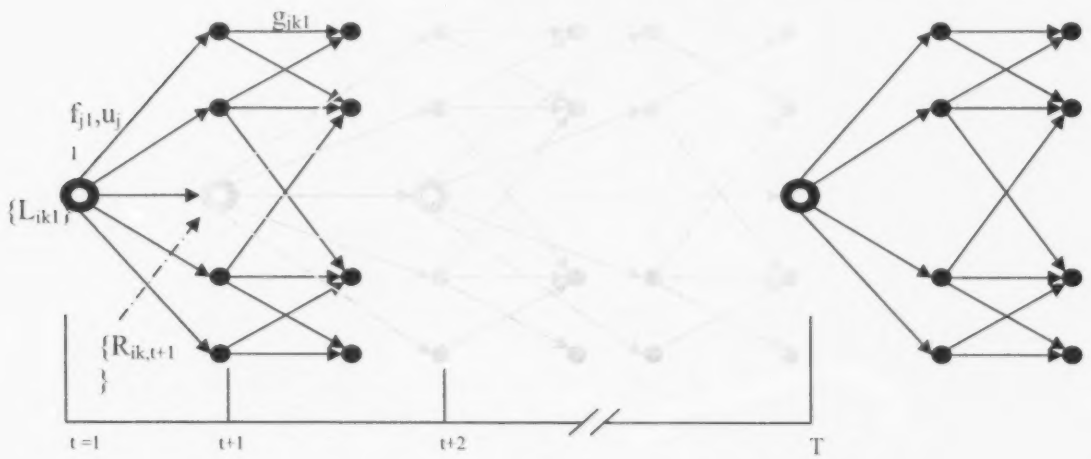

Figure 2 - The Single-Node Multiperiod Service Network Design Diagram

The deterministic multiperiod node service network design formulation may now be written as follows:

$$\text{Minimize } Z(x,y) = \sum_{t=1}^{T} \left[\sum_{j \in J_t} (f_{jt} y_{jt} + \sum_{k \in K_t} g_{jkt} x_{jkt}) + \sum_{k \in K_t} h_{kt} (R_{kt} - \sum_{j \in J_t} x_{jkt}) \right] \qquad (1)$$

$$\text{Subject to: } \sum_{k \in K_t} x_{jkt} \leq y_{jt} u_{jt} \qquad \forall j \in J_t, t = 1, \ldots, T \qquad (2)$$

$$\sum_{j \in J_t} x_{jkt} \leq R_{k,t-1} + \hat{R}_{kt} \qquad \forall k \in K_t, t = 1,\ldots,T \tag{3}$$

$$R_{kt} = R_{k,t-1} + \hat{R}_{kt} - \sum_{j \in J_t} x_{jkt} \qquad \forall k \in K_t, t = 1,\ldots,T \tag{4}$$

$$x_{jkt} \geq 0 \qquad \forall j \in J_t, k \in K_t, t = 1,\ldots,T \tag{5}$$

$$y_{jt} = \{0,1\} \qquad \forall j \in J_t, t = 1,\ldots,T \tag{6}$$

Equation (1) captures the cost of moving vehicles and shipments, as well as keeping shipments in storage, over the planning horizon. Equation (2) is the usual linking (or knapsack) restrictions on using only open services that also double as capacity constraints. Equation (3) is the demand constraints indicating the maximum flow possible out to a given terminal in any period, while equations (4) capture the dynamics of the system. Equations (5) and (6) are the usual non-negativity and integrality (on design decisions) constraints on decision variables.

The multi-period problem belongs to the family of fixed cost, capacitated, multicommodity network design problems that is known to be NP-hard and have proved difficult to solve for realistic problem instances. Consequently, heuristics are used in most cases and, as illustrated in Sections 2 and 3, this solution strategy is also used here.

Load arrivals are generally not deterministic, however. Consequently, we reformulate the problem to reflect the stochasticity of the demand and derive a solution strategy based on approximate dynamic programming techniques. The resulting procedure requires solving several instances of the deterministic single-period variant of formulation (1) – (6). Section 2 is dedicated to the presentation of this model and of the corresponding solution method.

1.2 The stochastic problem

Compared to the previous deterministic model, the new formulation considers explicitly the stochastic process of the demand arrivals. It is then transformed into a dynamic recursive model that introduces into the objective function a value that captures the impact of a holding decision over the planning horizon. The development follows and generalizes the work of Papadaki and Powell (2002, 2003).

Assume the arrival of the shipments follows a stochastic process with a general distribution. We assume for notational simplicity that arrivals are the only exogenous information process (our algorithmic strategy allows other sources of information). The problem consists of determining the dispatch policy that minimizes the total cost, that is, the policy that achieves the best trade-off between the dispatch and the inventory costs.

The stochastic formulation requires that we define the stochastic process for our arrivals. We are able to use a very general model. Let Ω be the set of elementary outcomes of the arrival process, with $\left(\hat{R}_t(\omega)\right)_{t=0}^{T}$ representing a sample realization of the number of arrivals. To define a formal probability space, we let $\mathcal{F}$ be the set of outcomes of $\hat{R}$ (more formally, the σ – algebra generated by $\hat{R}$) and let $\mathcal{P}$ be a probability measure on $(\Omega, \mathcal{F})$. Then $(\Omega, \mathcal{F}, \mathcal{P})$ is our probability space.

Let:

$C_t(x_t, y_t \mid R_t^+)$ = The one-period cost function given state R_t^+ :

$$= \sum_{j \in J_t} (f_{jt} y_{jt} + \sum_{k \in K_t} g_{jkt} x_{jkt}) + \sum_{k \in K_t} h_{kt} R_{kt} \tag{7}$$

Note that the cost function depends on the complete state variable R_t^+, but the holding costs are assessed only on the shipments left over at the end of the time period, given by R_t.

Our decisions (x,y) are made according to a *policy* π. We represent our *decision functions* using:

$X_t^\pi(R_t)$ = The flow decision function, which returns x at time t given state R_t.

$Y_t^\pi(R_t)$ = The dispatch decision function, which returns y at time t given state R_t

We let Π be our family of decision functions. Our objective function can now be formulated as:

$$F_t^\pi(R_t) = E\left\{\sum_{t'=t}^{T} C_{t'}(X_{t'}^\pi, Y_{t'}^\pi \mid R_{t'}^+)\right\}$$

The optimization problem, now, is to find the best policy, which is to say that we wish to solve:

$$F_0^* = \inf_{\pi \in \Pi}\left\{F_0^\pi(R_0)\right\}$$

Given the sheer complexity of solving this problem exactly, we resort to a search for "good" policies. We start by writing the problem in terms of the Bellman optimality equations. Given a policy π, each outcome $\omega \in \Omega$ produces a specific sequence of state variables $\left(R_t(\omega)\right)_{t=0}^{T}$. It is common in the dynamic programming community to define an *induced* stochastic process $\left(R_t(\omega)\right)_{t=0}^{T}$, with probability space $\left(\Omega, \mathcal{F}^\pi, \mathcal{P}^\pi\right)$. We use the standard notation $E^\pi\{\ \}$ to represent expectations over this induced probability space. We also let:

$V_t^\pi(R_t)$ = The future expected costs given that we start in period t in state R_t and follow policy π until the end of the horizon.

Since we are using an incomplete state variable, the value functions $V_t^\pi(R_t)$ are defined using a nonstandard form of the Bellman equations:

$$V_{t-1}^\pi(R_{t-1}) = E^\pi\left\{\min_{y_t,x_t} C_t(y_t,x_t \mid R_t^+) + V_t^\pi\left(R_t(y_t,x_t)\right) \middle| R_{t-1}\right\} \tag{8}$$

The value function V_t is too complex, however, and cannot be solved directly. The challenge is to find an approximation $\hat{V}_t(R_t)$ that is computationally tractable and still represents in a satisfactory way the impact of the decisions taken in the period of decision over time. Given an approximation $\hat{V}_t(R_t)$, we can take a Monte Carlo sample ω and find the decisions in period t using:

$$\left(Y_t^\pi(R_t,\omega), X_t^\pi(R_{t-1},\omega)\right) = \arg\min_{y_t,x_t} C_t\left(y_t,x_t \mid R_t^+(\omega)\right) + \hat{V}_t\left(R_t(y_t,x_t,\omega)\right) \tag{9}$$

We see, then, that an approximation $\hat{V}_t(R_t)$ determines a policy. Papadaki and Powell (2002, 2003) evaluate three types of approximations when R_t is a scalar: discrete, linear and nonlinear (concave). If R_t is a scalar, equation (8) may be solved optimally. The results show that the linear and nonlinear approximations converge much more quickly than the discrete approximation, with results within one to two percent of optimality. The nonlinear approximation outperforms the linear, but only after hundreds of iterations. The linear approximation works well because the function $\hat{V}_t(R_t)$ is monotone. Not surprisingly, the linear approximation works especially well when the underlying problem is stochastic. For our application, we will generally be able to run only a few dozen iterations, and hence the linear approximation offers not only simplicity, but also works the best among the approximations that have been tested.

Equation (9) also illustrates the importance of using the incomplete state variable for stochastic problems. When $\hat{R}_t$ is a scalar, the expectation in equation (8) is generally easy to compute. When $\hat{R}_t$ is a vector, the expectation is computationally intractable. In equation (9), we took a sample realization and computed $\left(Y_t^\pi(R_t,\omega), X_t^\pi(R_{t-1},\omega)\right) = \left(Y_t^\pi(R_t^+), X_t^\pi(R_t^+)\right)$, where the decisions are allowed to "see" $\hat{R}_t$. Had we used a complete state variable, the same trick would have required finding the decisions in period t allowing the decisions to see $\hat{R}_{t+1}$, which is a violation of the information constraint on the decision function. This trick of solving approximations of the Bellman equations by formulating the optimality equations around the incomplete state variable, combined with the use of appropriate functional forms for the value function, has been the basis of work on large-scale fleet management problems (Godfrey and Powell 2002a, 2002b) and multiproduct batch service problems (Papadaki and Powell 2003). The general idea is summarized in Powell and Topa-

loglu (2003). We note that for the special case of linear approximations, it is possible to use either the complete or incomplete variable (see Powell and Topaloglu, 2003, for a discussion of this point). Even though we are using linear approximations in this paper, we have adopted to use the incomplete state variable both for consistency with other papers on this topic, and to avoid a fundamental change in notation if other researchers wish to investigate other classes of approximations (any approximation other than linear requires converting to the incomplete state variable).

A linear approximation of the value function can be written:

$$\begin{aligned}\hat{V}_t(R_t,\omega) &= \hat{v}_t \bullet R_t \\ &= \sum_{k\in K_t} \hat{v}_{kt} R_{kt} \\ &= \sum_{k\in K_t} \hat{v}_{kt}\left(R_{k,t-1} + \hat{R}_{k,t}(\omega) - \sum_{j\in J_t} x_{jkt}\right) \qquad (10)\end{aligned}$$

Substituting (10) into (9) gives:

$$\begin{aligned}\left(Y_t^{\pi}(R_t,\omega), X_t^{\pi}(R_{t-1},\omega)\right) &= \arg\min_{y_t,x_t} C_t\left(y_t,x_t \mid R_t^{+}(\omega)\right) + \sum_{k\in K_t} \hat{v}_{kt}\left(R_{k,t-1} + \hat{R}_{k,t}(\omega) - \sum_{j\in J_t} x_{jkt}\right) \\ &= \arg\min_{y_t,x_t} C_t\left(y_t,x_t \mid R_t^{+}(\omega)\right) - \sum_{k\in K_t} \hat{v}_{kt}\left(\sum_{j\in J_t} x_{jkt}\right) + \sum_{k\in K_t} \hat{v}_{kt}\left(R_{k,t-1} + \hat{R}_{k,t}(\omega)\right) \qquad (11)\end{aligned}$$

The last term on the right hand side of (10) is a constant with respect to x and y, and hence can be ignored. The middle term is linear in x_t. Combining the definition of $C_t\left(y_t,x_t \mid R_t^{+}(\omega)\right)$ in equation (7) with equation (11), and dropping the constant term, gives:

$$\left(Y_t^{\pi}(R_t,\omega), X_t^{\pi}(R_{t-1},\omega)\right) = \arg\min_{y_t,x_t} \sum_{j\in J_t}(f_{jt}y_{jt} + \sum_{k\in K_t}(g_{jkt} - \hat{v}_{kt})x_{jkt}) + \sum_{k\in K_t} h_{kt}R_{kt} \qquad (12)$$

The challenge now is to devise a strategy to update v_t in such a way that the formulation for the dynamic problem can be solved satisfactorily. Since v_t is an estimate of the slope of Z_t we can define:

$$v_{kt} = \sum_{k \in K_t} \frac{\partial V_t}{\partial R_{kt}} \tag{13}$$

And, by using a finite difference approximation of the derivative, one obtains:

$$\tilde{v}_{kt} \cong [\tilde{Z}_t(R_t + \Delta R_{kt}) - \tilde{Z}_t(R_t)] / \Delta R_{kt} \tag{14}$$

The $\tilde{v}_{kt}$ slope captures the impact on the value of the objective function of having in inventory at the beginning of period t (before the arrival of the new shipments) one extra unit of flow with destination k. Using the definition of the value function $\tilde{Z}_t$, one obtains

$$\tilde{v}_{kt} = \Delta y_{jt} + \Delta x_{jkt} + \frac{\sum_{k \in K_t} (h_{kt} - v_{kt+1})}{\Delta R_{kt}} \left\{ (R_t + \Delta R_{kt} + \hat{R}_{kt} - \sum_{j \in J_t} x_{jkt}) - (R_t + \hat{R}_{kt} - \sum_{j \in J_t} x_{jkt}) \right\} \tag{15}$$

where:

$$\Delta y_{jt} = \frac{y_{jt}(R_{kt} + \Delta R_{kt}) - y_{jt}(R_{kt})}{\Delta R_{kt}}$$

and

$$\Delta x_{jkt} = \frac{x_{jkt}(R_{kt} + \Delta R_{kt}) - x_{jkt}(R_{kt})}{\Delta R_{kt}}$$

The solution algorithm is schematically displayed in Figure 3. Each iteration of the method is composed of two steps, a forward step and a backward step. In the forward step, the deterministic service selection and flow assignment problem will be solved from time $t = 0$ to the end of the planning horizon (t = T), using the meta-heuristic strategy detailed in next section. The backward step proceeds from the end of the planning horizon to its beginning, and computed the slope values. For each period, the amount of flow that stays in inventory for each destination is increased by one unit ($\Delta r_k = 1$) and the variation caused by this perturbation is computed using expression (15). The process is repeated until the stop criteria are reached. At each iteration n, new sets of flows (y_{jt}, x_{jkt}) and slopes $\tilde{v}_{kt}$ are obtained. To avoid that the slopes $\tilde{v}_{kt}$ bounce around between iterations, we smooth them out using:

$$\bar{v}_{kt}^{n+1} = (1-\gamma^n)\bar{v}_{kt}^{n} + \gamma^n \tilde{v}_{kt}^{n+1}, \; 0 \le \gamma^n \le 1. \tag{16}$$

Step 0 – Initialization

Set $\bar{v}_{kt}^{0} = 0$, for all k,t. Let $n = 1$, $t = 0$.

Step 1 – Arrivals

Determine a set of arrivals for each period using a general distribution (historic data and forecasts)

Step 2 - *Forward*

For $t \leftarrow$ 1 to T **do**

Solve the optimization problem at the terminal using the method of Section 3.

End-for

Step 3 – *Backward*

For $t \leftarrow$ T to 1 **do**

For $k \leftarrow$ 1 to K **do**

Create a perturbation Δr_{kt}.

Compute the slopes, $\tilde{v}_{kt}^{n}$, re-optimizing the system.

Remove the perturbation.

End-for

End-for

Step 4 – Smooth

Use (33) to smooth the slopes computed in the backward step

Step 5 – Stop

If the stop criteria are reached STOP, otherwise, set n $\leftarrow$ n + 1 and return to 2.

Figure 3 – Algorithm for the Stochastic Multiperiod Problem

2 DETERMINISTIC SINGLE-PERIOD, SINGLE-NODE PROBLEM

The general strategy to address the multiperiod formulations of the previous section decomposes the problem by period and solves a sequence of deterministic single-period, single-terminal network design problems. The problem belongs to the class of capacitated, multicommodity network design problems and is known to be NP-Hard. Moreover, it has to be solved repeatedly and a very efficient solution method is required. In fact, good solutions obtained very quickly are of greater interest than optimal ones obtained at greater time cost. Optimality in this sense is a bonus, not an objective *per se*. We propose therefore an efficient tabu search meta-heuristic procedure. We first state the mathematical formulation, followed by the description of the ejection chain neighborhood we propose, and, finally, we introduce the complete tabu search meta-heuristic.

2.1 The single-period formulation

The network of the static, single-period problem is rooted at node (terminal) i, in period t, and iteration n. It is denoted by $G_{it}^{n} = \left(N_{it}^{n}, A_{it}^{n} \right)$ and represents the decision of a carrier, given a moment in time and the availability of a certain type of information. Figure 4 illustrates the concept. To simplify the notation in this section, we drop the indices n, t, and i except when necessary to avoid ambiguities.

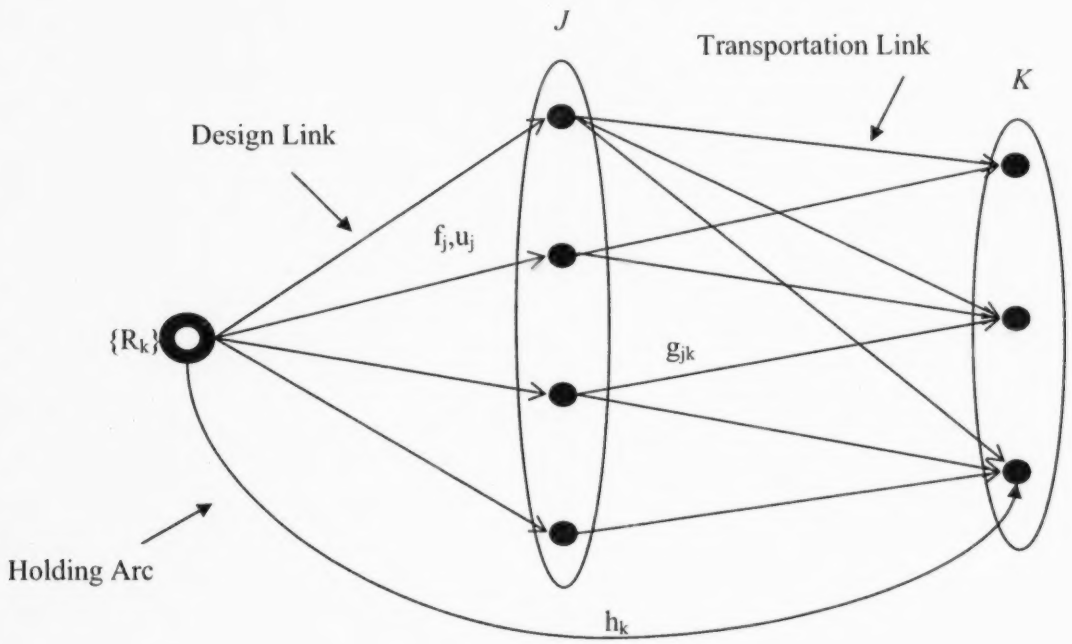

Figure 4 - Static, Single-Period Single-Node Service Network Design Problem

Following the notation of the previous section, demand is defined as the volume (number of shipments) R_k to be sent from the origin node to a destination $k \in K$. Nodes in K may be terminals or conceptual nodes created by aggregation. Let $L = \sum_{k \in K} R_k$ be the total demand at node i.

To reach a destination, a first movement (dispatch) is performed to one of a set of adjacent nodes $j \in J \subseteq N$. The root terminal is connected to the terminals in J by a set of design arcs (i,j) that represent the possible services at the terminal. From each $j \in J$, one can reach a destination $k \in K$ by using a transportation arc (j,k) that represents a direct link or a path linking the terminals. Transportation links capture the "future periods" effect of the multiperiod formulation. Thus, $N = \{i, J, K\}$ and $A = \{(i,j),(j,k) \mid j \in J, k \in K\}$.

A fixed cost f_j and a capacity u_j are associated to each design link. Only a cost g_{jk} is associated with transportation links, which represents the cost of sending one unit of flow to a destination k through an intermediate terminal j. When the intermediate node is the final destination, $g_{jj} = 0$. There is always the option of not moving the freight at this moment and this option implies a holding cost h_k. The decision variables represent whether or not a truck is to be dispatched to j (or, equivalently, the utilization of the intermediate terminals), and the routing of freight to each destination k:

- Design (dispatching) variables: y_j (y^n_{ijt}), $y_j = 1$ if node j is used (a vehicle is dispatched for i to j) and 0 otherwise. We assume in this paper that the length of the time period is set in such a way that only one departure is possible at each period.

- Freight flows: volumes sent from i to k through j: x_{jk} (x^n_{ijkt}).

The general formulation of Section 1 then becomes

$$\text{Minimize } Z(x,y) = \sum_{j\in J} (f_j y_j + \sum_{k\in K} g_{jk} x_{jk}) + \sum_{k\in K} h_k (R_k - \sum_{j\in J} x_{jk}) \qquad (17)$$

$$\text{Subject to: } \sum_{k\in K} x_{jk} \le y_j u_j \qquad \forall j \in J \qquad (18)$$

$$\sum_{j\in J} x_{jk} \le R_k \qquad \forall k \in K \qquad (19)$$

$$x_{jk} \ge 0 \qquad \forall j \in J, k \in K \qquad (20)$$

$$y_j = \{0,1\} \qquad \forall j \in J \qquad (21)$$

where constraints (18) are the usual linking (or knapsack) restrictions on using only available services, while equations (19) are flow (or demand) constraints.

Notice that for each destination k, one can explicitly enumerate all the paths from i to k through nodes j. These p_{ijk} paths are all made out of two arcs: $p_{ijk} = \{(i,j),(j,k)\}$. The cost of sending a full truck on each of the paths is given by $C_{p_{ijk}} = f_j + g_{jk} u_j$. It is then clear that for each destination k, there is a preferred route that ensures the smaller cost for moving a full truck. Therefore, when $R_k \ge u_j$, $\lfloor R_k / u_j \rfloor$ trucks will be dispatched on the preferred route and only routes for the remaining $R_k - \lfloor R_k / u_j \rfloor$ truck-equivalents will have to be determined. In the rest of this paper, we assume therefore that R_k represents this remaining quantity (and thus $R_k < u_j$).

Notice also that the objective function can be recast as:

$$Z = \sum_{j \in J} c_j y_j + \sum_{k \in K} \sum_{j \in J} (g_{jk} - h_k) x_{jk} + \sum_{k \in K} h_k R_k .$$

It is then clear that for all k such that $g_{jk} > h_k$, Z will be minimized for $x_{jk} = 0$. Thus, one has only to consider paths for which the transportation cost is less than the cost of "do nothing" (hold) option for the particular destination. It is also interesting to note that the path-based and the arc-based formulations of the network design model are the same, and that it may also be stated as a node design (location) model.

Three heuristics have been developed to address this problem and are compared in Section 3. All three are iterative improvement procedures based on moving through a search space defined by the continuous transportation variables. The first heuristic accepts only changes that improve the objective function. The other two are meta-heuristics based on the principles of tabu search to explore the neighborhood. These strategies are described in the following sections, following that of the neighborhood and the associated moves.

2.2 Ejection chain neighborhood and greedy descent procedure

The search strategies we examine proceed in the space of the continuous flow variables according to moves defined by *ejection chains*. An ejection chain is a neighborhood structure that combines simple moves to produce a composed movement. The neighbors of any given solution are obtained by a succession of moves, which are sequentially dependent. These neighbors are then evaluated, and the "best" is selected according to the particular heuristic procedure and becomes the current solution. See Glover and Laguna (1997) and Rego and Roucairol (1996) for a more detailed description of this approach.

In our case, the general idea is to realize interchanges between flows on design arcs. In certain cases, however, when flow is moved from one arc to another, the new arc may not have sufficient capacity to receive the new flow. It is then necessary to remove flow from this receiving arc and send it to other arcs. This process may be repeated after each interchange, creating a reaction chain of "receive flow and eject some other flow to make room." This means that each modification to the flow of one link implies a series of changes in the other links and freight volumes of the system. The objective of the procedure is to use these reaction chains in order to promote changes that produce a better solution.

Figure 5 illustrates the reaction caused by an interchange of flow between arcs. The upper left figure represents a feasible solution for the problem. The figures in the sequence show the reaction caused by a perturbation that moves a load from one service to another. The values between square parentheses represent the size of the shipments, while values between parentheses represent arc capacities. The reaction ends when a new feasible solution is reached. In order not to increase the computational times, an upper bound is usually imposed on the size of the ejection chain. Figure 6 displays the greedy strategy used to determine an ejection chain.

We use this neighborhood in a *Simple Heuristic* (*SH*) that greedily accepts an ejection chain that improves the current solution. Starting from an initial solution, the strategy can be described as follows. Determine an ejection chain using the procedure described above. If the new solution is better than the current solution, implement it; otherwise, determine a new ejection chain. The procedure

continues until a new solution is found. It is clear that the procedure will stop in the first local optimum encountered, irrespective of the quality of the solution. The tabu search meta-heuristics described next attempt to overcome this shortcoming.

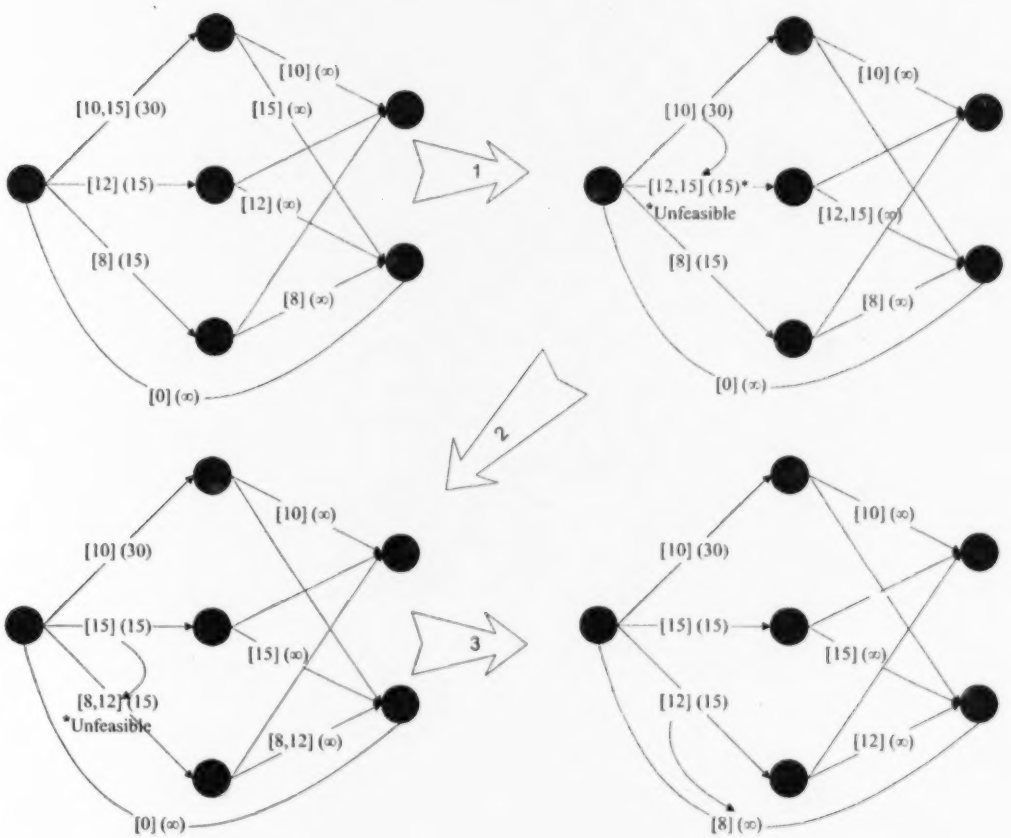

Figure 5 - Illustration of an Ejection Chain

```
Step 1 – Sort paths in increasing order of unit cost

Step 2 – Find a shipment that uses the highest unit cost
        Eject this shipment (c)
        Chain[1] ← c
        While (chain is not determined) or (maximum size of the chain is not reached))
        Do
            Find a new path for this shipment
            If (capacity ≥ size (c)) Then
                The chain is determined
            Else
                Find a flow to be removed (ejected) from the service
                Eject this flow (f)
                j ← j + 1
                Chain(j) ← (f)
            End-If
        End-while
        If (chain reached the maximum size) Then
            Put the last ejected flow on an inventory arc
        End-If
End
```

Figure 6 – Greedy Algorithm to Determine an Ejection Chain

2.3 Tabu Search Meta-heuristics

Tabu search is a meta-heuristic that has been successfully used to address many hard, combinatorial optimization problems, including network design (Glover 1986; Glover and Laguna 1997). Tabu search is an iterative procedure that uses memories of solutions (or attributes thereof) already visited to learn about the solution space and to guide the search out of local optima and towards promising regions. One or several neighborhoods are defined, together with the corresponding moves. Then, at each iteration, solutions neighboring the current solution are identified and the "best", not necessarily improving, one is selected as the new current solution. A short-term memory, usually identified as the *tabu list*, is used to avoid cycling by forbidding moves to solutions recently visited or sharing attributes with such solutions. When the highest solution quality possible is sought after, more advanced memory and neighborhood structures are advantageously used. In the present case, however, speed is of essence and we will not implement these features. Experimental results show, however, that the simple tabu search strategy we propose still achieves good quality solutions.

The tabu search procedures presented in the following are based on the neighborhood defined by the ejection chain move defined previously. Solutions are evaluated using the objective function of the design problem, which is also the aspiration criteria (i.e., a move to a tabu solution is accepted if the new solution has lower objective function value than the current best). The induced neighborhood is huge, however, since, in theory, one should construct all ejection chains, of all lengths, before selecting the solution to move to. This is impractical in any circumstance. We thus adopt a well-known technique in meta-heuristics in general and tabu search in particular and use a candidate list

A *candidate list* is a collection of neighbors among which the selection of the next solution is performed. We define two tabu search procedures that differ in how the candidate list is built: *Search First Feasible Chain* (*SFFC*) and *Search First Improving Chain* (*SFIC*).

The SFFC strategy builds ejection chains starting from the current solution and accepts the first feasible solution that either is not tabu or fulfills the aspiration criterion. Notice that non improving solutions are accepted. Therefore, if after a certain number of iterations, neither the best solution nor the current solution are improved, the procedure returns to the current best solution and proceeds with the search. If the search returns to the same solution several consecutive times, the search is stopped. The full description of the procedure appears in the Appendix.

A broader part of the neighborhood is explored by the SFIC procedure (see the Appendix for a detailed presentation). Here, ejection chains are built until a feasible solution is found that improves the current solution. If no chain improves the current solution, the one that least degrades the solution is implemented. If, after a certain number of iterations the current solution is still not improved, the procedure returns to the best current solution and the search continues on the second best chain for that solution. The process continues until it returns a certain number of times to the same best solution.

Irrespective of the strategy selected, an initial feasible solution has to be identified. The heuristic we use starts by sorting all paths in increasing order of cost. It then proceeds iteratively through the shipments and attempts to send every load on its shortest (cheapest) path. The next path in the list is used when the capacity of the path is reached. This heuristic is easy to implement and fast. Moreover, as illustrated by the computational results of the next section, the tabu search procedure obtains good solutions in short computational times.

Shipments are assigned to and moved among services based on the path valuations that reflect both the fixed and the variable costs of the system. The fixed service cost has to be paid as soon as the service is selected. In order not to penalize the shipments that are assigned first to a previously unselected service, approximations are used. Thus, to compute the initial solution, we associate the fixed cost to the capacity of the service, which yields the following path cost:

$$c_{p_{ijk}} = \frac{f_j}{u_j} + g_{jk}$$

Once the initial solution is found, the unit cost of path using selected services is computed in the usual way:

$$c_{p_{ijk}} = \frac{f_j}{x_j} + g_{jk}$$

On the other hand, unused arcs are "closed" ($y_{ij} = 0$) during the search. Then, to evaluate the cost of sending a load on a path that makes use of a not yet selected service, the corresponding arc is given a cost equal to the maximum cost value computed for that arc during the previous iterations, including those of the initial solution.

2.4 Long-term memory

As already noticed, the single-period, single-terminal problem has to be solved repeatedly when DSND problems are contemplated: at each time period and forward-backward iteration. This may be extremely time-consuming as illustrated in Section 3. Moreover, it is felt that the behavior of a terminal, the number of potential service departures, as well as the composition and amplitude of demand and, thus, the actual dispatch decisions, will not vary widely from period to period or from one iteration to the next. Some "order" exists in the randomness of demand arrivals and the operations of the system. Observations of the operations of actual transportation systems tend to support this hypothesis. Then, if such a behavioral pattern could be identified for a terminal, it could be used to accelerate the search by, for example, identifying the preferred service for shipments departing to certain final destinations. This should help to reduce significantly the computational time.

Our strategy is based on long-term memories that record, in a *frequency matrix*, the frequencies of services used to serve each demand to its final destination. This matrix is built during the execution of the algorithm. Once the static problem is solved for the current period, the matrix is updated by increasing the frequencies of the services used. After a few iterations, the dispatch pattern for each demand should start to emerge as the services most heavily used. We will use this pattern to bias the selection of services when initial solutions need to be computed. This approach is somewhat similar to the *target analysis* methodology proposed initially by Glover (see Glover and Laguna 1997).

3 Experimental Results

The experimental phase of our work was planned to examine several issues:

1. What is the performance in terms of computing efficiency and solution quality of the heuristics for the static, single-period problem? Which one is better? Recall that this problem has to be solved repeatedly when the dynamic problem is addressed, and that the number of repetitions is expected to get huge when full dynamic service network design problems are contemplated.

2. How efficiently can dynamic problems be solved? Is the long-term memory mechanism (the matrix of frequencies) helpful?

3. What is gained by explicitly considering the dynamics of the system compared to simply solving a series of single-period static instances?

3.1 Experimental Design

We tested the models on problem instances based on data from a large U.S. LTL motor carrier, operating numerous breakbulk and end-of-line terminals that serve the continental United States. We used this information to set the general parameters of our problem, but used randomly generated data so that we could control the characteristics of individual problem instances.

Three classes of problems were generated, each with a different ratio between demand and capacity. Class I problems have more demand than the capacity of the network, while the contrary is true for problems of Class III. Problem instances in Class II display a balance between demand and capacity. Problem instances were generated randomly, by using uniform distributions on the intervals presented in Table 1 for capacity and demand, as well as for service (fixed), transportation, and holding costs. Problems of four different sizes, small, medium, large, and extra large, were generated as defined in Table 2. Dynamic problem instances were generated following a similar process but generating a different vector of demand for each period of a 7-day planning horizon.

Table 1 – General Characteristics of Test Problem Instances

Problem Class	Capacity	Demand	Fixed Cost	Holding Cost	Transportation Cost
Class I	[50,150]	[150,300]	[50,300]	[100,300]	[20,120]
Class II	[150,400]	[130,260]	[50,300]	[100,300]	[20,120]
Class III	[150,450]	[30,60]	[50,300]	[100,300]	[20,120]

Table 2 – Test Problem Instances Dimensions

Problem Size	# of Terminals	# of Intermediate Terminals	# of Intermediate for each Destination
Small	[10,20]	[3,8]	[2,5]
Medium	[21,100]	[10,50]	[4,12]
Large	[101,250]	[50,100]	[8,20]
XLarge	[260,450]	[100,200]	[16,50]

For the sake of conciseness and clarity, only aggregated results are included in the paper. Detailed results are presented in Dall'Orto (2001).

3.2 Static Experiments

As indicated previously, the solution procedure for the single-period, single-terminal service design problem must be very efficient. Of course, the quality of the solution is also of major importance. A number of parameters characterize the heuristics and impact their performance. These parameters must be calibrated prior to undertaking the performance analysis.

3.1 Experimental Design

We tested the models on problem instances based on data from a large U.S. LTL motor carrier, operating numerous breakbulk and end-of-line terminals that serve the continental United States. We used this information to set the general parameters of our problem, but used randomly generated data so that we could control the characteristics of individual problem instances.

Three classes of problems were generated, each with a different ratio between demand and capacity. Class I problems have more demand than the capacity of the network, while the contrary is true for problems of Class III. Problem instances in Class II display a balance between demand and capacity. Problem instances were generated randomly, by using uniform distributions on the intervals presented in Table 1 for capacity and demand, as well as for service (fixed), transportation, and holding costs. Problems of four different sizes, small, medium, large, and extra large, were generated as defined in Table 2. Dynamic problem instances were generated following a similar process but generating a different vector of demand for each period of a 7-day planning horizon.

Table 1 – General Characteristics of Test Problem Instances

Problem Class	Capacity	Demand	Fixed Cost	Holding Cost	Transportation Cost
Class I	[50,150]	[150,300]	[50,300]	[100,300]	[20,120]
Class II	[150,400]	[130,260]	[50,300]	[100,300]	[20,120]
Class III	[150,450]	[30,60]	[50,300]	[100,300]	[20,120]

Table 2 – Test Problem Instances Dimensions

Problem Size	# of Terminals	# of Intermediate Terminals	# of Intermediate for each Destination
Small	[10,20]	[3,8]	[2,5]
Medium	[21,100]	[10,50]	[4,12]
Large	[101,250]	[50,100]	[8,20]
XLarge	[260,450]	[100,200]	[16,50]

For the sake of conciseness and clarity, only aggregated results are included in the paper. Detailed results are presented in Dall'Orto (2001).

3.2 Static Experiments

As indicated previously, the solution procedure for the single-period, single-terminal service design problem must be very efficient. Of course, the quality of the solution is also of major importance. A number of parameters characterize the heuristics and impact their performance. These parameters must be calibrated prior to undertaking the performance analysis.

The calibration phase has been conducted using 250 problem instances (small, medium, and large of all types), on a PC equipped with a Pentium III processor with an 800 MHz clock. Lindo 6.1 was used to obtain the optimal solutions.

Five parameters characterize the heuristics. The values tested for each one of them were:

1. *Maximum number of iterations* the procedure was allowed to run (if not stopped otherwise). We tested values equal to 100, 200, 300, and 500 for HS, and 500, 1000, 1500, 2000, 2500, and 3500 for the CFFC and CFIC tabu search procedures.

2. *Maximum size of the ejection chain.* This parameter limits the number of load exchanges allowed when ejection chains are built. Values equal to 4, 6, 8 , 10, and 12 were tested.

3. *Tabu list size* that indicates the number of iterations a solution is tagged tabu, e.g., one cannot return to it unless it satisfies the aspiration criteria (is better than the current best). This parameter attempts to avoid cycling and to foster a broader exploration of the solution space. Tabu list length of 5, 10, 15, and 20 were tested.

4. *Maximum number of returns* to the same solution before stopping the procedure. When this limit is attained, it is an indication that, given the settings of the other parameters, the search has reached a local optimum from which it is incapable to escape. We tested 4, 6, 8, 10, and 12.

5. *Maximum number of successive iterations without improvement.* When the search reaches this limit, it returns to the current best solution to initiate a new search thread. We tested 3, 5, and 10.

The procedures and parameter settings were compared according to the percentage of deviation from the optimal solution and the computation time required. The general conclusion of these experiments is that the procedures are robust with respect to the parameter settings. For each heuristic, the results for different parameter settings are very similar, the calibration appearing more as fine tuning of parameters, rather than a discrimination procedure.

As expected, tabu search outperforms greedy descent in solution quality, the procedure using First Improvement (SFIC) reaching higher quality solutions than the one (SFFC) that moves to the First Feasible solution. Again, without surprise, the relation on computing times displays exactly the opposite characteristics. In fact, the number of iterations has no impact on the performance of the greedy HS heuristic, provided it is sufficiently long to allow the procedure to reach a local optimum. This is not true for the tabu search procedures that generally obtain better solutions when allowed to search longer. Yet, the rate of improvement slows down after a certain point that we fixed empirically to 2500 iterations.

We somehow expected to find a relation between the length of the ejection chains and the problem dimensions. We did not find such a relation in our experiments, however. We rather identified a relation between the ejection chain length and the type of solution method.

The other three parameters do not apply to the HS procedure. For tabu search, the duration of tabu tags is generally an important parameter. This turned out to be the case here as well, the tabu

list length impacting both the solution quality and the computational times. The last 2 parameters, the maximum number of returns to the same solution and the maximum number of successive iterations without improvement, did not impact the solution quality in any significant way. They did impact computing times, though. The final parameter settings appear in Table 3 (an X indicates that the parameter is not used by the procedure) and were used to perform the comparison analysis of the three heuristics present next.

A new set of 100 problems of all types and dimensions were generated for the comparative analysis of the proposed procedures. We generated a larger number of Class II problems, where demand and capacity are balanced, because they appear more difficult to solve than Class I (the solution consists in using "all" services and holding the excess demand) and Class III (where the preferred service is used for almost all demands) problem instances.

Table 3 – The Parameters for Each Solution Strategy

Parameter	SH	SFFC	SFIC
Max. iterations	200	2500	2500
Max. chain size	6	8	8
Tabu list size	X	15	10
Max. returns to the same solution	X	8	4
Max. iterations without improvement	X	10	10

Optimal solutions have been obtained by using CPLEX 7.1. All experiments have been performed on a SUN Workstation. The results are summarized in Tables 4, 5, and 6. We indicate average results for the percentage of optimality reached (a value of 100 indicates that optimal solutions have been found), as well as for the computational time in seconds CPU (the figures are given within square brackets).

Two main conclusions emerge from the experimental results summarized in these tables. First, despite its apparent simplicity, the single-node, single-period service network design problem is difficult to solve optimally within a short computation time for most problems of interest. Even for the easy problem instances of Class I, where demand exceeds supply, the exact method required significantly more time than the heuristics. Second, the heuristics we propose achieve good solutions both in terms of deviation from the optimal solution and in computation effort required to achieve these results.

Table 4 – Results for Class I (Optimum Percentage and Time [sec])

Problem Size				
Small	100 [0.015]	100 [0.0325]	100 [0.0325]	100 [0.0325]
Medium	100[0.245]	100 [0.085]	100 [0.265]	100 [0.110]
Large	100[8.455]	100 [0.508]	100 [1.886]	100 [0.851]
XLarge	100[58.852]	99.98[0.587]	99.98[3.538]	99.98[2.658]

Table 5 – Results for Class II (Optimum Percentage and Time [sec])

Problem Size		SH	SFFC	SFIC
Small	100 [0.0233]	89.37 [0.04]	99.11 [0.04]	96.79 [0.04]
Medium	100 [0.48]	88.03 [0.0525]	92.20 [0.4025]	92.28 [0.2803]
Large	100 [12.05]	84.88 [0.85]	89.39 [2.25]	90.18 [3.30]
XLarge	100[68.75]	85.14[1.50]	90.54[5.05]	91.54[8.52]

Table 6 – Results for Class III (Optimum Percentage and Time [sec])

Problem Size	CPLEX	SH	SFFC	SFIC
Small	100 [0.01]	96.65 [0,1775]	98.61 [0.0425]	98.84 [0.0325]
Medium	100 [0.265]	93.94 [0.33]	96.71 [0.365]	97.89 [0.1083]
Large	100 [9.343]	88.31 [0.466]	92.49 [1.995]	95.04 [2.852]
XLarge	100 [64.26]	88.46 [0.978]	92.47 [3.678]	92.80 [4.162]

Comparing the performances of the three heuristics, it is again clear that the greedy heuristic is the fastest but yields the poorer results. Tabu search obtains the best solutions, the First Improvement strategy offering the best performances in this respect at the expense of slightly longer computational times. With respect to the type of instance, problems are easy when demand exceeds supply. The simple heuristic is sufficient in these cases. An excess of capacity (Class III) also makes problems easier to address, although not as easy as for Class I instances. It is for problems of Class II,

list length impacting both the solution quality and the computational times. The last 2 parameters, the maximum number of returns to the same solution and the maximum number of successive iterations without improvement, did not impact the solution quality in any significant way. They did impact computing times, though. The final parameter settings appear in Table 3 (an X indicates that the parameter is not used by the procedure) and were used to perform the comparison analysis of the three heuristics present next.

A new set of 100 problems of all types and dimensions were generated for the comparative analysis of the proposed procedures. We generated a larger number of Class II problems, where demand and capacity are balanced, because they appear more difficult to solve than Class I (the solution consists in using "all" services and holding the excess demand) and Class III (where the preferred service is used for almost all demands) problem instances.

Table 3 – The Parameters for Each Solution Strategy

Parameter	SH	SFFC	SFIC
Max. iterations	200	2500	2500
Max. chain size	6	8	8
Tabu list size	X	15	10
Max. returns to the same solution	X	8	4
Max. iterations without improvement	X	10	10

Optimal solutions have been obtained by using CPLEX 7.1. All experiments have been performed on a SUN Workstation. The results are summarized in Tables 4, 5, and 6. We indicate average results for the percentage of optimality reached (a value of 100 indicates that optimal solutions have been found), as well as for the computational time in seconds CPU (the figures are given within square brackets).

Two main conclusions emerge from the experimental results summarized in these tables. First, despite its apparent simplicity, the single-node, single-period service network design problem is difficult to solve optimally within a short computation time for most problems of interest. Even for the easy problem instances of Class I, where demand exceeds supply, the exact method required significantly more time than the heuristics. Second, the heuristics we propose achieve good solutions both in terms of deviation from the optimal solution and in computation effort required to achieve these results.

Table 4 – Results for Class I (Optimum Percentage and Time [sec])

Problem Size	CPLEX	SH	SFFC	SFIC
Small	100 [0.015]	100 [0.0325]	100 [0.0325]	100 [0.0325]
Medium	100[0.245]	100 [0.085]	100 [0.265]	100 [0.110]
Large	100[8.455]	100 [0.508]	100 [1.886]	100 [0.851]
XLarge	100[58.852]	99.98[0.587]	99.98[3.538]	99.98[2.658]

Table 5 – Results for Class II (Optimum Percentage and Time [sec])

Problem Size	CPLEX	SH	SFFC	SFIC
Small	100 [0.0233]	89.37 [0.04]	99.11 [0.04]	96.79 [0.04]
Medium	100 [0.48]	88.03 [0.0525]	92.20 [0.4025]	92.28 [0.2803]
Large	100 [12.05]	84.88 [0.85]	89.39 [2.25]	90.18 [3.30]
XLarge	100[68.75]	85.14[1.50]	90.54[5.05]	91.54[8.52]

Table 6 – Results for Class III (Optimum Percentage and Time [sec])

Problem Size	CPLEX	SH	SFFC	SFIC
Small	100 [0.01]	96.65 [0,1775]	98.61 [0.0425]	98.84 [0.0325]
Medium	100 [0.265]	93.94 [0.33]	96.71 [0.365]	97.89 [0.1083]
Large	100 [9.343]	88.31 [0.466]	92.49 [1.995]	95.04 [2.852]
XLarge	100 [64.26]	88.46 [0.978]	92.47 [3.678]	92.80 [4.162]

Comparing the performances of the three heuristics, it is again clear that the greedy heuristic is the fastest but yields the poorer results. Tabu search obtains the best solutions, the First Improvement strategy offering the best performances in this respect at the expense of slightly longer computational times. With respect to the type of instance, problems are easy when demand exceeds supply. The simple heuristic is sufficient in these cases. An excess of capacity (Class III) also makes problems easier to address, although not as easy as for Class I instances. It is for problems of Class II,

representative of most actual cases where capacity "balances" demand, that the differences among heuristics and the advantages of the tabu search appear clearly.

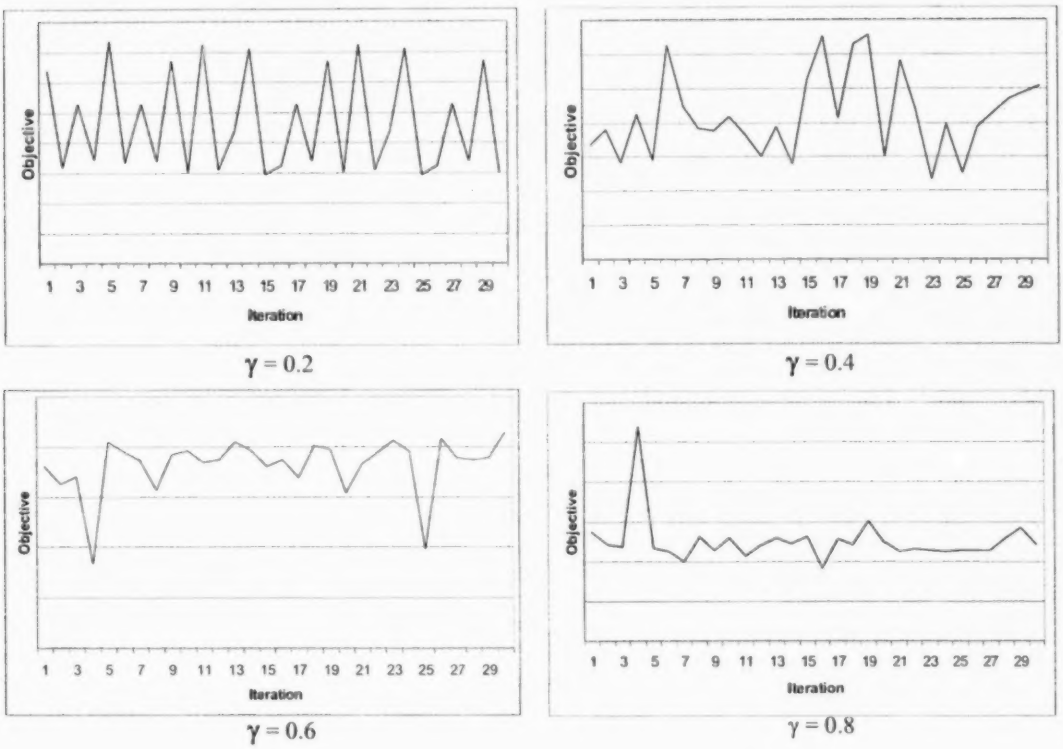

Figure 7 – Convergence Behavior According to the Smoothing Factor

3.3 – Dynamic Experiments

The only calibration required for the dynamic, multi-period algorithm concern the parameters normally used to ensure convergence. Thus, for example, Powell and Carvalho (1998) use an upper bound on the empty movements and a smoothing factor to guarantee the convergence of their strategy. We decided not to impose limits on the quantities kept in inventory to avoid making some problems infeasible. Thus, we use the smoothing factor only, even though it does not guarantee the convergence of the dynamic model. We tested using a smoothing factor varying from 0.2 to 0.8 (see Figure 7). The best behavior was observed using $\gamma = 0.8$: with this value the results oscillate around the best solution found.

This setting was then used to analyze the performance of the dynamic algorithm. Notice that the first forward step in the dynamic strategy uses slopes equal to zero. This is the same thing as solving the dynamic problem using the static strategy sequentially and independently for each decision period. Thus, the performance of the dynamic approach is measured as the improvement it in-

duces compared to the initial, greedy-myopic solution. 84 new test problems have been generated for these experiments, 21 instances for each problem size.

Table 7 displays the average improvement percentages induced by using the dynamic algorithm for these test problems. The SH strategy does not improve a (good) greedy-myopic initial solution. It proves to be incapable of escaping the corresponding local optimum. The tabu search procedures obtain more significant improvements, especially for problems of class II. For classes I and III one observes, again, that there is little to improve when there is almost no demand or capacity to manage. For problems of class II, where capacity is sufficient but not in excess and good operating policies are important, the tabu search methods obtain very good results, with an average improvement of 15.55% and 17.19% for the SFFC and SFIC strategies, respectively. It is important to emphasize that class II problems are the most common in practice.

Table 7 – Improvement Yield by the Dynamic Algorithm from a Greedy-Myopic Solution

Problem Class	SH	SFFC	SFIC
I	0	1.04	0.46
II	0.06	15.55	17.19
III	0	0.54	1.43

The computational time required by a straightforward application of the dynamic procedure appears too large, however, of the order of 1 to 3 minutes. The factor that most contributes to these times is the backward pass where the problem is solved several times to compute the marginal values. Most of these problems are very similar, differing by one unit of demand only. We therefore repeated the experiments using the long-term memory mechanism described at Section 2.4. Table 8 presents the aggregated results.

Table 8 – Comparison of Computational Time for the Dynamic Algorithm

	Greedy-Myopic	Dynamic SFFC	Dynamic SHIC
Total time "straight"	11.5	940.6	817.4
Total time "memory"	11.5	190.8	177.8
Gain	-	79.72%	78.25%

Each of the first two rows of Table 8 displays the total CPU time required to apply the algorithm indicated in each column to all 84 problem instances without ("straight") and with ("memory") the long-term memory. The last row indicates the gain (in percentage) obtained by using the frequency matrix memory. The results are given for the greedy-myopic method and for the two dynamic algorithms that correspond to using each of the two tabu search methods to solve the single-period problem. These results confirm the importance of the long-term memory for the dynamic solution method and opens up interesting perspectives for addressing full scale DSND problems.

duces compared to the initial, greedy-myopic solution. 84 new test problems have been generated for these experiments, 21 instances for each problem size.

Table 7 displays the average improvement percentages induced by using the dynamic algorithm for these test problems. The SH strategy does not improve a (good) greedy-myopic initial solution. It proves to be incapable of escaping the corresponding local optimum. The tabu search procedures obtain more significant improvements, especially for problems of class II. For classes I and III one observes, again, that there is little to improve when there is almost no demand or capacity to manage. For problems of class II, where capacity is sufficient but not in excess and good operating policies are important, the tabu search methods obtain very good results, with an average improvement of 15.55% and 17.19% for the SFFC and SFIC strategies, respectively. It is important to emphasize that class II problems are the most common in practice.

Table 7 – Improvement Yield by the Dynamic Algorithm from a Greedy-Myopic Solution

Problem Class	SH	SFFC	SFIC
I	0	1.04	0.46
II	0.06	15.55	17.19
III	0	0.54	1.43

The computational time required by a straightforward application of the dynamic procedure appears too large, however, of the order of 1 to 3 minutes. The factor that most contributes to these times is the backward pass where the problem is solved several times to compute the marginal values. Most of these problems are very similar, differing by one unit of demand only. We therefore repeated the experiments using the long-term memory mechanism described at Section 2.4. Table 8 presents the aggregated results.

Table 8 – Comparison of Computational Time for the Dynamic Algorithm

	Greedy-Myopic	Dynamic SFFC	Dynamic SHIC
Total time "straight"	11.5	940.6	817.4
Total time "memory"	11.5	190.8	177.8
Gain	-	79.72%	78.25%

Each of the first two rows of Table 8 displays the total CPU time required to apply the algorithm indicated in each column to all 84 problem instances without ("straight") and with ("memory") the long-term memory. The last row indicates the gain (in percentage) obtained by using the frequency matrix memory. The results are given for the greedy-myopic method and for the two dynamic algorithms that correspond to using each of the two tabu search methods to solve the single-period problem. These results confirm the importance of the long-term memory for the dynamic solution method and opens up interesting perspectives for addressing full scale DSND problems.

4 – Conclusions

We have examined the single-node dynamic service network design problem and presented two formulations: a time-dependent, stochastic formulation that aims to optimize the problem over a given planning horizon, and a static, single-period, model that appears as a subproblem when addressing both the time-dependent version and general DSND cases. Appropriate solution methods have been presented for both formulations. In particular, a tabu search framework based on ejection chains has been introduced for the single-period problem. Moreover, a long-term learning mechanism has been introduced as well, which allows significant gains in computing efficiency for the dynamic algorithm. Experimentations performed on a large number of problem instances derived from actual data indicate that the methods presented are both computationally very efficient and provide solutions close to the optimal ones. This opens very promising perspectives for the general dynamic service network design problem.

Acknowledgments

Partial funding for this project has been provided by the Natural Sciences and Engineering Research Council of Canada, the Fonds F.C.A.R. of the Province of Québec (Canada), and CNPq - The National Research Council of Brazil. The first author (Dall'Orto) thanks CAPES (Brazil) for the PhD fellowship he received as well as for the financial support to the visit to Canada. The work of the last author (Powell) was partially supported by grant AFOSR-F49620-93-1-0098 from the Air Force Office of Scientific Research.

References

Balakrishnan, A., Magnanti, T.L., and Mirchandani, P. (1997). Network Design. In Dell'Amico, M., Maffioli, F., and Martello, S., editors, *Annotated Bibliographies in Combinatorial Optimization*, pages 311-334, John Wiley & Sons, New York, NY.

Barnhart, C., Jin, H., and Vance, P.H. (2000). Railroad Blocking: A Network Design Application. *Operations Research*, 48(4):603-614.

Barnhart, C. and Schneur, R.R. (1996). Network Design for Express Freight Service, *Operations Research*, 44(6):852-863

Bertazzi, L. and Speranza, M.G. (1999). Inventory Control on Sequences of Links with Given Transportation Frequencies. *International Journal of Production Economics*, 56:261-270.

Bertazzi, L. and Speranza, M.G. (2002). Continuous and Discrete Shipping Strategies for the Single Link Problem. *Transportation Science*, 36(3):314-325.

Blumenfeld, D.E., Burns, L.D., Diltz, J.D., and Daganzo, C.F. (1985). Analyzing Trade-Off Between Transportation, Inventory and Production Costs on Freight Networks. *Transportation Research*, 19B(5):361-380.

Burns, L.D., Hall, R.W., Blumenfeld, D.E., and Daganzo, C.F. (1984). Distributions Strategies that Minimize Transportation and Inventory Cost. *Operations Research*,

33(3):469-490.

Cordeau, J-F., Toth, P., and Vigo, D. (1998). A Survey of Optimization Models for Train Routing and Scheduling. *Transportation Science*. 32(4):380-404.

Crainic, T. G. (2000). Network Design in Freight Transportation. *European Journal of Operational Research*, 122(2):272-288.

Crainic, T. G. (2003). Long-Haul Freight Transportation. In R. Hall, editor, *Handbook of Transportation Science*, 2nd Edition, pages 451-516, Kluwer Academic Publishers, Norwell, MA.

Crainic, T. G. and G. Laporte (1997). Planning Models for Freight Transportation. *European Journal of Operational Research*, 97(3):409-434.

Crainic, T. G. and J. Roy (1988). OR Tools for Tactical Freight Transportation Planning. *European Journal of Operational Research*. 33:290-297.

Crainic, T. G. and J-M. Rousseau (1986). Multicommodity, Multimode Freight Transportation: A General Modeling and Algorithmic Framework for the Service Network Design Problem. *Transportation Research B: Methodology*, 20(B):225-242.

Crainic, T. G., J-A. Ferland and J-M Rousseau (1984). A Tactical Planning Model for Rail Freight Transportation. *Transportation Science*, 18(2):165-184.

Daganzo, C. F. (1991). *Logistics Systems Analysis*. Springer-Verlag, Berlin.

Dall'Orto, L.C. (2001). *Modelagem e algorithmos para o Problema do Projeto de Redes de Serviço Dinâmicas*. Ph.D. Thesis, Department of Industrial Engineering, Pontificia Universidade Catolica do Rio de Janeiro, Brazil.

Deb, R. and Serfozo, R. (1973), Optimal Control of Batch Service Queues, *Advances in Applied Probability*, 5:340-361.

Equi, L., Gallo, G., Marziale, S., and Weitraub, A. (1997). A Combined Transportation and Scheduling Problem. *European Journal of Operational Research*, 97(1):94-104.

Farvolden, J. and Powell, W.B. (1994). Subgradient Methods for the Service Network Design Problem. *Transportation Science*, 28(3):256-272.

Glover, F. (1986). Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research*, 13:533-549.

Glover, F. and Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers, Norwell, MA.

Godfrey, G.A. and Powell, W.B. (2002a). An Adaptive Dynamic Programming Algorithm for Dynamic Fleet Management I: Single period Travel Times. *Transportation Science*, 36(1):40-54.

Godfrey, G.A. and Powell, W.B. (2002b). An Adaptive Dynamic Programming Algorithm for Dynamic Fleet Management II: Multi-Period Travel Times. *Transportation Science*, 36(1):21-39.

Gorman, M.F. (1998). An Application of Genetic and Tabu Search to the Freight Railroad Operating Plan Problem. *Annals of Operations Research*, 78:51-69.

Grünert, T. and Sebastian, H.-J. (2000). Planning Models for Long-Haul Operations of Postal and Express Shipment Companies. *European Journal of Operational Research*, 122:289-309.

Haghani, A.E. (1989). Formulation and Solution of Combined Train Routing and Makeup, and Empty Car Distribution Model. *Transportation Research B*, 23B(6):433-452.

Ignall, E. and Kolesar, P. (1972). Operating Characteristics of a Simple Shuttle under Local Dispatching Rules, *Operations Research,* 20:1077-1088.

Kim, D., Barnhart, C., Ware, K., and Reinhardt, G. (1999). Multimodal Express Package Delivery: A Service Network Design Application. *Transportation Science*, 33(4):391-407.

Kosten, L. (1967). The Custodian Problem. in *Queueing Theory: Recent Developmentst and Applications*, English Press, Ltd., London, pp. 65-70.

Kosten, L. (1973). *Stochastic Theory of Service Systems*, Pergamon Press, New York.

Magnanti, T.L. and Wong, R.T. (1986). Network Design and Transportation Planning: Models and Algorithms, *Transportation Science*, 18(1):1-55.

Minoux, M. (1986). Network Synthesis and Optimum Network Design Problems: Models, Solution Methods and Applications. *Networks*, 19:313-360.

Papadaki, K. and Powell, W.B. (2002). Exploiting Structure in Adaptive Dynamic Programming Algorithms for a Stochastic Batch Service Problem. *European Journal of Operational Research,* 142(1):108-127.

Papadaki, K. and Powell, W.B. (2003). An Adaptive Dynamic Programming Algorithm for a Stochastic Multiproduct Batch Dispatch Problem, *Naval Research Logistics* (to appear).

Pflug, G. (1996). *Optimization of Stochastic Models: the Interface Between Simulation and Optimization*, Kluwer Academic Publishers, Norwell, MA.

Powell, W.B. (1986). A Local Improvement Heuristic for the Design of Less-than-Truckload Motor Carrier Networks. *Transportation Science*, 20(4):246-357.

Powell, W.B. and Carvalho, T.A. (1997). Dynamic Control of Multicommodity Fleet Management Problems. *European Journal of Operational Research,* 98:522-541.

Powell, W.B. and Carvalho, T.A. (1998). Dynamic Control of Logistics Queueing Networks for Large-Scale Fleet Management. *Transportation Science*, 32(2):90-109.

Powell, W.B. and Humblet, P. (1986). The Bulk Service Queue with a General Control Strategy: Theoretical Analysis and a New Computational Procedure, *Operations Research*, 34:267-275.

Powell, W.B. and Sheffi, Y. (1989). Design and Implementation of an Interactive Optimization System for Network Design in the Motor Carrier Industry, *Operations Research*, 37(1):12-29 (1989).

Powell, W. B. and Topaloglu, H. (2003) Stochastic Programming in Transportation and Lo-

gistics, in Shapiro, A. and Ruszczynski, A., editors, *Handbooks in Operations Research and Management Science: Stochastic Programming*, (to appear).

Rego, C. and Roucairol, C. (1996). A Parallel Tabu Search Algorithm Using Ejection Chains for the VRP. In Osman, I.H. and Kelly, J.P., editors, *Meta-Heuristics: Theory & Applications*, pages 253-295, Kluwer Academic Publishers, Norwell, MA.

Speranza, M.G. and Ukovich, W. (1992). A Decision Support System for Materials Management. *International Journal of Computer Integrated Manufacturing*, 5:239-244.

Speranza, M.G. and Ukovich, W. (1994). Minimizing Transportation and Inventory Costs for Several Products on a Single Link, *Operations Research*, 42:879-894.

Speranza, M.G. and Ukovich, W. (1996). An Algorithm for Optimal Shipments with Given Frequencies. *Naval Research Logistics*, 46:399-417.

Tsitsiklis, J. and van Roy, B. (1997). An Analysis of Temporal-Difference Learning with Function Approximation, *IEEE Transactions on Control*, 42:674-690.

APPENDIX

Here follow the detailed procedural descriptions of the two tabu search procedures. Let X represent a solution (y,x) to the single-node service network design problem and Z(X) the corresponding value of the objective function.

The SFFC Strategy

Step 0 – Initialize the tabu list
Set X_{best}, $X_{actual} \leftarrow X_0$
Return_best, Chain_search $\leftarrow 0$

Step 1 – Determine an ejection chain from the current solution
Compute the variation due to this ejection chain

Step 2 – **If** (chain is not tabu) **or** (chain is tabu but satisfies aspiration criterion) **Then**
X_{actual}, *Tabu_list* $\leftarrow X$
If $(Z(X)>Z(X_{actual})$ **Then** *Chain_search* $\leftarrow$ *Chain_search* $+1$
If $(Z(X)<Z(X_{best})$ **Then**
$X_{best} \leftarrow X$
Return_best, Chain_search $\leftarrow 0$
End if
If (*Chain_search* > *max_search*) **Then**
$X_{actual} \leftarrow X_{best}$
Return_best $\leftarrow$ *Return_best* + 1
End if
End Else
If (*Return_best* > *max_retorn*) **Then**
Go **End** (the search is finished)
End Else
Go to Step 1
End if
End-if
End

The SFIC Strategy

Step 0 – Initialize the tabu list
Set X_{best}, $X_{actuale}$, $X_{bast_actual} \leftarrow X_0$
Retorn_best, Return_chain, Chain_search ← 0

Step 1 – Determine an ejection chain for the actual solution
Compute the variation caused by this ejection chain

Step 2 – **If** (chain is not tabu) **or** (is tabu but satisfies aspiration criterion) **Then**
IF ($Z(X) < Z(X_{actual})$) **then**
X_{best_actual}, Xactual, *tabu_list* ← X
Retorn_chain, Chain_search ← 0
If ($Z(X) < z(X_{best})$) **then**
$X_{best} \leftarrow X$
Retorn_best ← 0
End if
Return to Step 1
Else
If (chain is better then the current chain that degrades the solution the least) **then**
$X_{degrade} \cdot X$
Return to Step 1
End if
End if
If (a chain has not been found that improves the current solution) **Then**
$X_{actual} \leftarrow X_{degrrade}$
Chain_search ← *Chain_search* + 1
If (*Chain_search* > *max_search*) **then**
$X_{actual} \leftarrow X_{bset_actual}$
Return_chain ← *Return_chain* + 1
If (*Return_chain* > *max_retorn*) **then**
$X_{actual} \leftarrow X_{best}$
Return_best ← *Return_best* + 1
If (*Return_best* > *max_retorn*) **then**
Go to the **End**
Else Return to Step 1
Else Return to Step 1
End if
Else Return to Step 1
End if
End if
End